



软件科学基础

MORESTLC: More on the Simply Typed Lambda-Calculus

熊英飞
北京大学



扩展STLC

- STLC目前只有布尔类型和基本函数调用
- 加入更多语言成分
 - 自然数
 - Let
 - Pairs
 - Unit
 - Sums
 - Lists
 - 递归调用
 - Records



自然数

- 直接将Types部分定义的自然数语法、语义和类型规则加入即可



Let

let $x = 1+5$ in

let $y = x + 1$ in

$y + 2$



Let

Syntax:

$t ::=$	Terms
...	(other terms same as before)
let $x=t$ in t	let-binding

Reduction:

$$\frac{t_1 \rightarrow t_1'}{\text{let } x=t_1 \text{ in } t_2 \rightarrow \text{let } x=t_1' \text{ in } t_2} \text{ (ST_Let1)}$$

$$\frac{}{\text{let } x=v_1 \text{ in } t_2 \rightarrow [x:=v_1]t_2} \text{ (ST_LetValue)}$$

Typing:

$$\frac{\Gamma \vdash t_1 \in T_1 \quad x \mapsto T_1; \Gamma \vdash t_2 \in T_2}{\Gamma \vdash \text{let } x=t_1 \text{ in } t_2 \in T_2} \text{ (T_Let)}$$

Unit



Syntax:

t ::=	Terms
...	(other terms same as before)
unit	unit

v ::=	Values
...	
unit	unit value

T ::=	Types
...	
Unit	unit type

Typing:

$$\frac{}{\Gamma \vdash \text{unit} \in \text{Unit}} \text{ (T_Unit)}$$



命令作为项

- 没有返回值的命令可以认为返回Unit
 - $t1 := t2 : \text{Unit}$
- 命令的序列可以看做函数调用的简写
 - $t1; t2$ 等价于 $(\lambda x:\text{Unit}, t2) t1$
- 之后在Reference章节会进一步学习



Pairs – 示例

```
\x : Nat*Nat,  
  let sum = x.fst + x.snd in  
  let diff = x.fst - x.snd in  
  (sum, diff)
```



Pairs-语法

$t ::=$	
...	
(t_1, t_2)	pair
$t_0.fst$	first projection
$t_0.snd$	second projection

$v ::=$	
...	
(v_1, v_2)	pair value

$T ::=$	
...	
$T_1 * T_2$	product type

Pairs-语义



$$\frac{t_1 \rightarrow t_1'}{(t_1, t_2) \rightarrow (t_1', t_2)} \quad (\text{ST_Pair1})$$

$$\frac{t_2 \rightarrow t_2'}{(v_1, t_2) \rightarrow (v_1, t_2')} \quad (\text{ST_Pair2})$$

$$\frac{t_0 \rightarrow t_0'}{t_0.\text{fst} \rightarrow t_0'.\text{fst}} \quad (\text{ST_Fst1})$$

$$\overline{(v_1, v_2).\text{fst} \rightarrow v_1} \quad (\text{ST_FstPair})$$

$$\frac{t_0 \rightarrow t_0'}{t_0.\text{snd} \rightarrow t_0'.\text{snd}} \quad (\text{ST_Snd1})$$

$$\overline{(v_1, v_2).\text{snd} \rightarrow v_2} \quad (\text{ST_SndPair})$$



Pairs-类型

$$\frac{\text{Gamma} \vdash t_1 \in T_1 \quad \text{Gamma} \vdash t_2 \in T_2}{\text{Gamma} \vdash (t_1, t_2) \in T_1 * T_2} \quad (\text{T_Pair})$$

$$\frac{\text{Gamma} \vdash t_0 \in T_1 * T_2}{\text{Gamma} \vdash t_0.\text{fst} \in T_1} \quad (\text{T_Fst})$$

$$\frac{\text{Gamma} \vdash t_0 \in T_1 * T_2}{\text{Gamma} \vdash t_0.\text{snd} \in T_2} \quad (\text{T_Snd})$$



Records – 示例

```
\x: {age:Nat, sex:Bool},  
  if x.age > 18 then tru else fls
```



Records-语法

$t ::=$	
...	
$\{i_1=t_1, \dots, i_n=t_n\}$	record
$t.i$	projection
$v ::=$	
...	
$\{i_1=v_1, \dots, i_n=v_n\}$	record value
$T ::=$	
...	
$\{i_1:T_1, \dots, i_n:T_n\}$	record type



Records-语义

$$\frac{ti \rightarrow ti'}{\{i_1=v_1, \dots, i_m=v_m, in=ti, \dots\} \rightarrow \{i_1=v_1, \dots, i_m=v_m, in=ti', \dots\}} \quad (\text{ST_Rcd})$$

$$\frac{t_0 \rightarrow t_0'}{t_0.i \rightarrow t_0'.i} \quad (\text{ST_Proj1})$$

$$\frac{}{\{\dots, i=v_i, \dots\}.i \rightarrow v_i} \quad (\text{ST_ProjRcd})$$



Records

$$\frac{\text{Gamma} \vdash t_1 \in T_1 \quad \dots \quad \text{Gamma} \vdash t_n \in T_n}{\text{Gamma} \vdash \{i_1=t_1, \dots, i_n=t_n\} \in \{i_1:T_1, \dots, i_n:T_n\}} \quad (\text{T_Rcd})$$

$$\frac{\text{Gamma} \vdash t_0 \in \{\dots, i:T_i, \dots\}}{\text{Gamma} \vdash t_0.i \in T_i} \quad (\text{T_Proj})$$



Records 可以表示为 Pair 和 Unit

- `{age=5, sex=tru}`
表示为
`(5, (tru, unit))`
- 确实有编译器是这样实现 Record 的，不过更常见的是用偏移量



Sum-示例

$\text{div} \in \text{Nat} \rightarrow \text{Nat} \rightarrow (\text{Nat} + \text{Unit})$

$\text{div} =$

$\lambda x:\text{Nat}, \lambda y:\text{Nat},$

if iszero y then

inr Nat unit

else

inl Unit (x / y)



Sum-语法

$t ::=$	Terms
...	(other terms same as before)
$\text{inl } T_2 \ t_1$	tagging (left)
$\text{inr } T_1 \ t_2$	tagging (right)
case t_0 of	case analysis
$\text{inl } x_1 \Rightarrow t_1$	
$\text{inr } x_2 \Rightarrow t_2$	

$v ::=$	Values
...	
$\text{inl } T_2 \ v_1$	tagged value (left)
$\text{inr } T_1 \ v_2$	tagged value (right)

$T ::=$	Types
...	
$T_1 + T_2$	sum type



Sum-语义

$$\frac{t_1 \rightarrow t_1'}{\text{inl } T_2 \ t_1 \rightarrow \text{inl } T_2 \ t_1'} \quad (\text{ST_Inl})$$

$$\frac{t_2 \rightarrow t_2'}{\text{inr } T_1 \ t_2 \rightarrow \text{inr } T_1 \ t_2'} \quad (\text{ST_Inr})$$

$$\frac{t_0 \rightarrow t_0'}{\text{case } t_0 \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \rightarrow \text{case } t_0' \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2} \quad (\text{ST_Case})$$

$$\frac{}{\text{case (inl } T_2 \ v_1) \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \rightarrow [x_1 := v_1]t_1} \quad (\text{ST_CaseInl})$$

$$\frac{}{\text{case (inr } T_1 \ v_2) \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \rightarrow [x_2 := v_2]t_2} \quad (\text{ST_CaseInr})$$



Sum-类型

$$\frac{\text{Gamma} \vdash t_1 \in T_1}{\text{Gamma} \vdash \text{inl } T_2 \ t_1 \in T_1 + T_2} \quad (\text{T_Inl})$$

$$\frac{\text{Gamma} \vdash t_2 \in T_2}{\text{Gamma} \vdash \text{inr } T_1 \ t_2 \in T_1 + T_2} \quad (\text{T_Inr})$$

$$\frac{\begin{array}{l} \text{Gamma} \vdash t_0 \in T_1 + T_2 \\ x_1 \mapsto T_1; \text{Gamma} \vdash t_1 \in T_3 \\ x_2 \mapsto T_2; \text{Gamma} \vdash t_2 \in T_3 \end{array}}{\text{Gamma} \vdash \text{case } t_0 \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \in T_3} \quad (\text{T_Case})$$



Variant

- 同Record类似，Sum也可以扩展为Variant
 - `<some:Nat, none:unit>`
- Variant和Record合用，可以起到Rocq中Inductive定义的作用

```
Inductive rgb : Type :=  
  | red  
  | green  
  | blue.  
Inductive color : Type :=  
  | black  
  | white  
  | primary (p : rgb).
```

```
<black:unit,  
white:unit,  
primary: {p:<red:unit,  
           green:unit,  
           blue:unit}>>>
```

- 练习：用Variant定义natlist



List

- 在支持Universal Type和Recursive Type的编程语言中，List可以定义为用户定义类型
- 本课程不涉及以上两种类型，所以将List定义为语言扩展

```
Inductive list (X:Type) : Type :=  
  | nil  
  | cons (x : X) (l : list X).
```



List-语法

```
t ::=                               Terms
| ...
| nil T                             empty list
| t1 :: t2                         cons
| case t1 of                         case analysis
    nil => t2
    | xh::xt => t3
```

```
v ::=                               Values
| ...
| nil T                             nil value
| v1 :: v2                         cons value
```

```
T ::=                               Types
| ...
| List T                             list of Ts
```



List-语义

$$\frac{t_1 \rightarrow t_1'}{t_1 :: t_2 \rightarrow t_1' :: t_2} \text{ (ST_Cons1)}$$

$$\frac{t_2 \rightarrow t_2'}{v_1 :: t_2 \rightarrow v_1 :: t_2'} \text{ (ST_Cons2)}$$

$$\frac{t_1 \rightarrow t_1'}{(\text{case } t_1 \text{ of nil } \Rightarrow t_2 \mid \text{ xh}::\text{xt} \Rightarrow t_3) \rightarrow (\text{case } t_1' \text{ of nil } \Rightarrow t_2 \mid \text{ xh}::\text{xt} \Rightarrow t_3)} \text{ (ST_Lcase1)}$$

$$\frac{}{(\text{case nil } T_1 \text{ of nil } \Rightarrow t_2 \mid \text{ xh}::\text{xt} \Rightarrow t_3) \rightarrow t_2} \text{ (ST_LcaseNil)}$$

$$\frac{}{(\text{case } (v\text{h}::v\text{t}) \text{ of nil } \Rightarrow t_2 \mid \text{ xh}::\text{xt} \Rightarrow t_3) \rightarrow [\text{xh}:=v\text{h}][\text{xt}:=v\text{t}]t_3} \text{ (ST_LcaseCons)}$$



List-类型

$$\frac{}{\text{Gamma} \vdash \text{nil } T_1 \in \text{List } T_1} \text{ (T_Nil)}$$

$$\frac{\text{Gamma} \vdash t_1 \in T_1 \quad \text{Gamma} \vdash t_2 \in \text{List } T_1}{\text{Gamma} \vdash t_1 :: t_2 \in \text{List } T_1} \text{ (T_Cons)}$$

$$\frac{\begin{array}{c} \text{Gamma} \vdash t_1 \in \text{List } T_1 \\ \text{Gamma} \vdash t_2 \in T_2 \\ (\text{xh} \mapsto T_1; \text{xt} \mapsto \text{List } T_1; \text{Gamma}) \vdash t_3 \in T_2 \end{array}}{\text{Gamma} \vdash (\text{case } t_1 \text{ of nil} \Rightarrow t_2 \mid \text{xh}::\text{xt} \Rightarrow t_3) \in T_2} \text{ (T_Lcase)}$$



复习：Y组合子

$$Y = \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$$

- $Y f = f (Y f)$

撰写递归程序的基本手段



递归

- Y组合子可以在lambda演算中定义，但其类型是递归类型，本课程不涉及
 - 作为语言成分定义
- 首先撰写函数把自己作为参数传入
 - $\text{fact} = \backslash \text{self: Nat} \rightarrow \text{Nat},$
 $\backslash x: \text{Nat},$
 if $x=0$ then 1 else $x * (\text{self} (\text{pred } x))$
- 然后定义高阶函数负责传入“自己”，即起Y组合子的作用
 - $\text{fix fact: Nat} \rightarrow \text{Nat}$



递归

Syntax:

$t ::=$	Terms
...	
$\text{fix } t_1$	fixed-point operator

Reduction:

$$\frac{t_1 \rightarrow t_1'}{\text{fix } t_1 \rightarrow \text{fix } t_1'} \quad (\text{ST_Fix1})$$

$$\frac{}{\text{fix } (\lambda x f:T_1. t_1) \rightarrow [\lambda x f:=\text{fix } (\lambda x f:T_1. t_1)] t_1} \quad (\text{ST_FixAbs})$$

Typing:

$$\frac{\Gamma \vdash t_1 \in T_1 \rightarrow T_1}{\Gamma \vdash \text{fix } t_1 \in T_1} \quad (\text{T_Fix})$$



Progress和Preservation

- 二者在扩展后的STLC上仍然成立

```
Theorem progress : forall t T,  
  empty |-- t \in T ->  
  value t \/ exists t', t --> t'.
```

```
Theorem preservation : forall t t' T,  
  empty |-- t \in T ->  
  t --> t' ->  
  empty |-- t' \in T.
```

- 具体证明留作作业



作业

- 完成MoreSTLC中standard非optional的6道习题